

INFORMATICA

Información práctica:

PRÓXIMO JUEVES, SEGUNDO PARCIAL EN HORARIO DE CLASE
08:30 – 10:30

AVISARÉ POR MOODLE DEL AULA DE REALIZACIÓN DEL EXAMEN

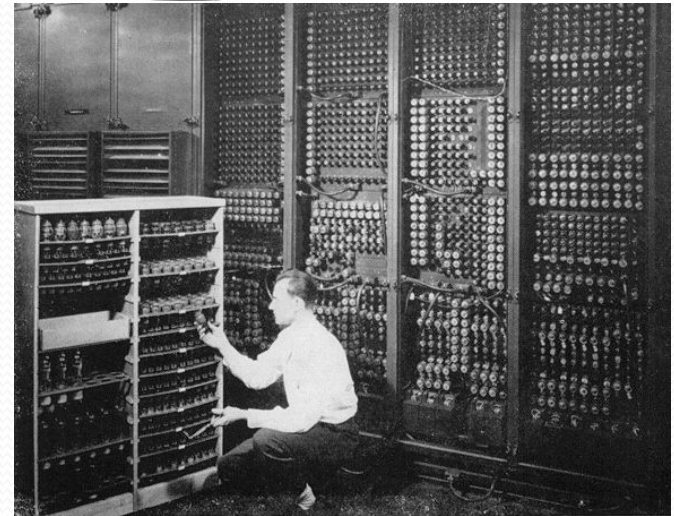
EXAMEN CON ORDENADOR

SE PUEDE UTILIZAR TODO EL MATERIAL QUE QUERÁIS (CÓDIGO, APUNTES...)

MUY RECOMENDABLE TENER PROGRAMADOS (Y SABER PROGRAMAR) LOS EJERCICIOS PROPUESTOS EN MOODLE (CARPETA MATERIAL ADICIONAL DE ESTUDIO)

Debugging

- ¿Qué es un bug?
- Es un error o fallo en un programa
- El Mark II, construido en 1944, sufrió un fallo en un relé electromagnético. Cuando se investigó se vió que una polilla se había colado dentro y provocaba que ese relé quedase abierto



9/9

0800 Machine started
1000 stopped - machine ✓ { 1.2700 9.032 847 025
13'00 MP - MC 1.482 1.0000 9.017 846 895 correct
033 PRO 2 2.130476495 4.615 925057(-12)
check 2.130476495
Relays 6-2 in 033 failed speed test 11.00 test.
in turn (Relays changed)

1100 Started Cosine Tape (Sine check)
1525 Started Multiplier Adder Test.

1545 Relay #70 Panel F (moth) in relay.

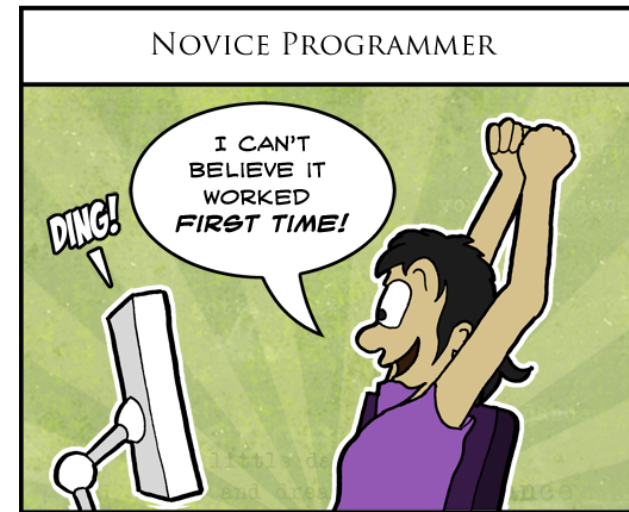
First actual case of bug being found.
1630 Machine started.
1700 closed down.

Relay 2145
0209 3376

Debugging

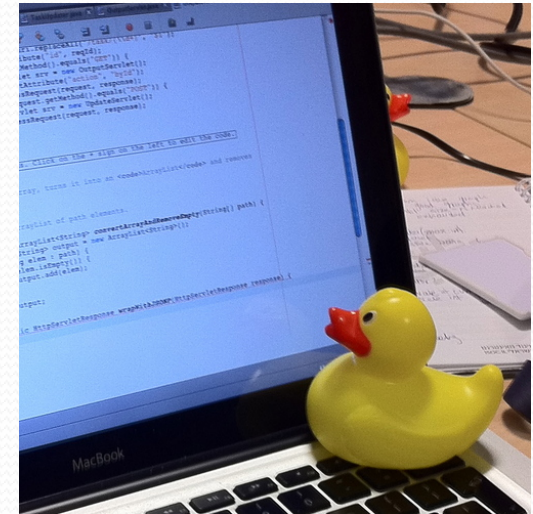
- **debugging** is the process of locating and fixing or bypassing bugs (errors) in computer program code or the engineering of a hardware device. To **debug** a program or hardware device is to start with a problem, isolate the source of the problem, and then fix it.
- FORTRAN

```
WRITE(*,*) "variable =", variable  
READ(*,*)
```



Debugging

- Rubber duck debbuging
 - The name is a reference to a story in the book *The Pragmatic Programmer* in which a programmer would carry around a rubber duck and debug their code by forcing themselves to explain it, line-by-line, to the duck.^[1]
- Off by one (obi-wan) error
 - A loop of some sort in which the index is off by one.



https://en.wikipedia.org/wiki/Rubber_duck_debugging

<http://home.agh.edu.pl/~szymon/jargon/html/O/obi-wan-error.html>

Debugging

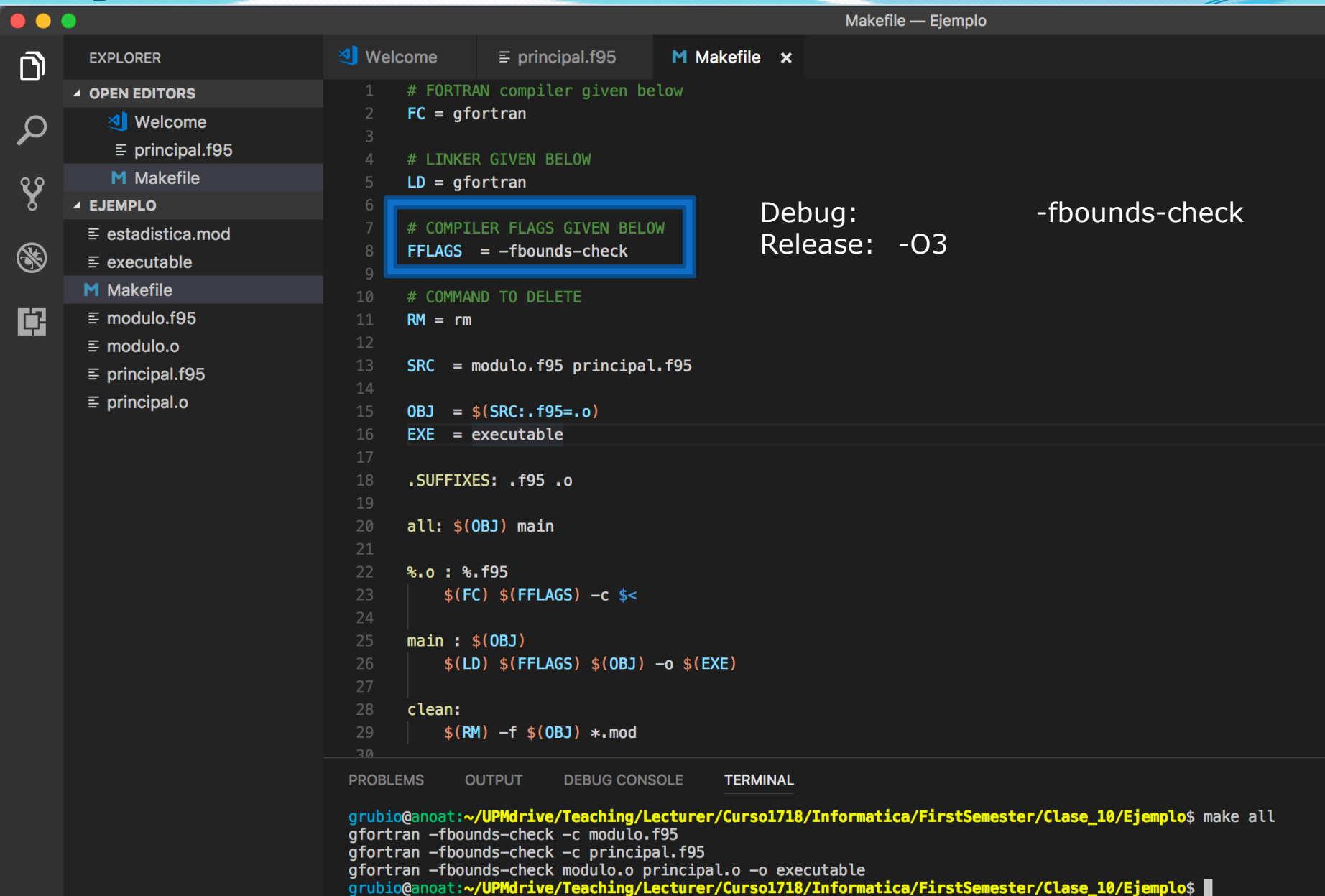
- **Flags**

- -fbounds-check: Check if the array index is within the bounds.
- -O0 -O1, -O2, -O3: Different level of optimization. Increase the speed of execution

- GFORTRAN info

- <https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html>
- http://faculty.washington.edu/rjl/uwamath583s11/sphinx/notes/html/gfortran_flags.html
- <http://stackoverflow.com/questions/3676322/what-flags-do-you-set-for-your-gfortran-debugger-compiler-to-catch-faulty-code>

Programación modular: makefile



The image shows a Visual Studio Code window titled "Makefile — Ejemplo". The Explorer sidebar on the left shows a project structure with files like `estadistica.mod`, `executable`, `modulo.f95`, `modulo.o`, `principal.f95`, and `principal.o`. The Makefile editor is open, showing the following content:

```
1  # FORTRAN compiler given below
2  FC = gfortran
3
4  # LINKER GIVEN BELOW
5  LD = gfortran
6
7  # COMPILER FLAGS GIVEN BELOW
8  FFLAGS = -fbounds-check
9
10 # COMMAND TO DELETE
11 RM = rm
12
13 SRC = modulo.f95 principal.f95
14
15 OBJ = $(SRC:.f95=.o)
16 EXE = executable
17
18 .SUFFIXES: .f95 .o
19
20 all: $(OBJ) main
21
22 %.o : %.f95
23 |     $(FC) $(FFLAGS) -c $<
24
25 main : $(OBJ)
26 |     $(LD) $(FFLAGS) $(OBJ) -o $(EXE)
27
28 clean:
29 |     $(RM) -f $(OBJ) *.mod
30
```

A blue box highlights the line `FFLAGS = -fbounds-check` on line 8. To the right of the editor, the text "Debug: -fbounds-check" and "Release: -O3" is displayed.

The TERMINAL panel at the bottom shows the execution of the `make all` command:

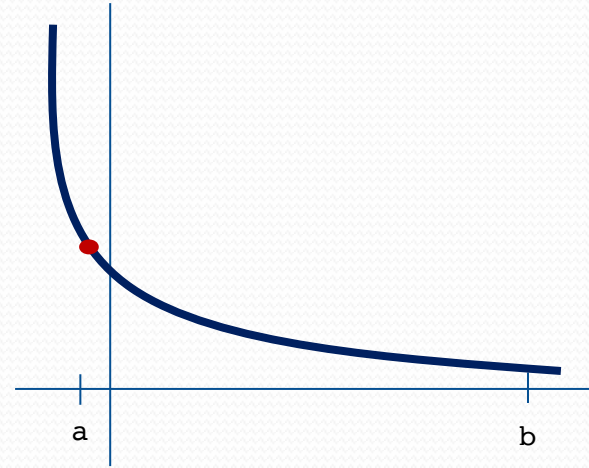
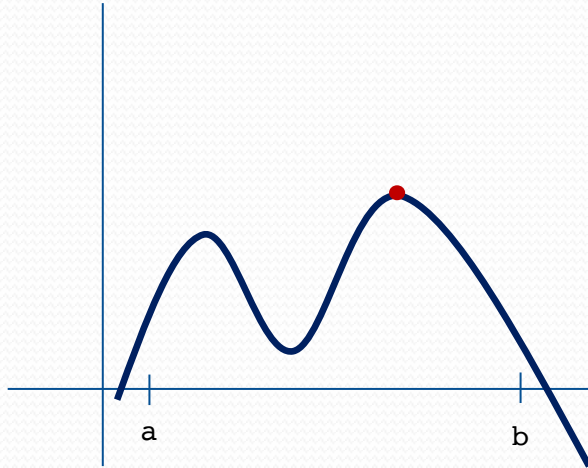
```
grubio@anoat:~/UPMdrive/Teaching/Lecturer/Curso1718/Informatica/FirstSemester/Clase_10/Ejemplo$ make all
gfortran -fbounds-check -c modulo.f95
gfortran -fbounds-check -c principal.f95
gfortran -fbounds-check modulo.o principal.o -o executable
grubio@anoat:~/UPMdrive/Teaching/Lecturer/Curso1718/Informatica/FirstSemester/Clase_10/Ejemplo$
```

Programación modular: Avanzada

Ejemplo

Ejemplo

- Subrutina que, dado un intervalo $[a,b]$, devuelve el valor máximo que alcanza una función arbitraria (dato de entrada) dentro de dicho intervalo.

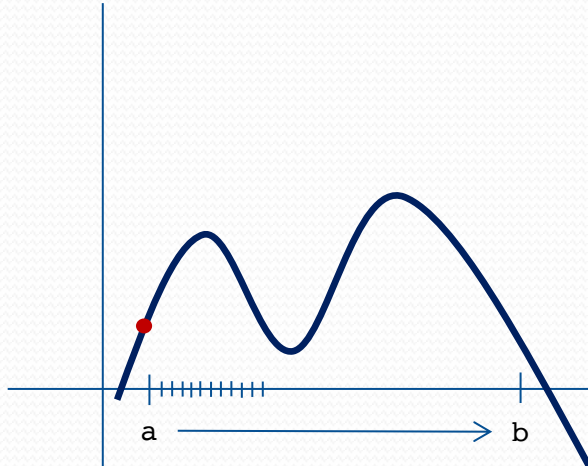


Programación modular: Avanzada

Ejemplo

Algoritmia:

La forma en que lo he pensado consiste en ‘barrer’ el intervalo y evaluar la función en cada punto.



`valor maximo = f(a)`

`Si $f(x) > \text{valor maximo} \Rightarrow \text{valor maximo} = f(x)$`
 `$x \in [a, b]$`

Partición o discretización del intervalo

Programación modular: Avanzada

Implementación:

```
module funciones
```

```
contains
```

```
function f1(x)  
  real :: x  
  real :: f1
```

```
    f1 = exp(x*2)
```

```
end function
```

```
function f2(x)  
  real :: x  
  real :: f2
```

```
    f2 = cos(x)+sin(x)
```

```
end function
```

```
function f3(x)  
  real :: x  
  real :: f3
```

```
    f3 = x*x + 1
```

```
end function
```

```
end module
```

```
subroutine valor_maximo_intervalo(x0, x1, func, valor_max)
```

```
  real, intent(in) :: x0,x1
```

```
  real, intent(inout) :: valor_max
```

```
end subroutine
```

Programación modular: Avanzada

Implementación:

```
module funciones
```

```
contains
```

```
function f1(x)  
  real :: x  
  real :: f1
```

```
  f1 = exp(x*2)
```

```
end function
```

```
function f2(x)  
  real :: x  
  real :: f2
```

```
  f2 = cos(x)+sin(x)
```

```
end function
```

```
function f3(x)  
  real :: x  
  real :: f3
```

```
  f3 = x*x + 1
```

```
end function
```

```
end module
```

```
subroutine valor_maximo_intervalo(x0, x1, func, valor_max)
```

```
  real, intent(in)      :: x0,x1  
  real, intent(inout)   :: valor_max
```

```
  interface
```

```
    function func(x)
```

```
      real :: x
```

```
      real :: func
```

```
    end function
```

```
  end interface
```

```
  ! Variables locales
```

```
end subroutine
```

Programación modular: Avanzada

Implementación:

```
subroutine valor_maximo_intervalo(x0, x1, func, valor_max)
```

```
    real, intent(in)          :: x0,x1  
    real, intent(inout)       :: valor_max
```

```
    interface  
        function func(x)  
            real    :: x  
            real    :: func  
        end function  
    end interface
```

```
    ! Variables locales
```

El uso de la construcción interface sirve para declarar subprogramas, funciones o subrutinas, como argumentos de otros subprogramas.

Sólo hace falta indicar el número, orden y tipo de las variables que usa el subprograma que se pasa como argumento.

```
end subroutine
```

Programación modular: Avanzada

Implementación:

```
subroutine valor_maximo_intervalo(x0, x1, func, valor_max)
```

```
    real, intent(in)          :: x0,x1  
    real, intent(inout)       :: valor_max
```

```
    interface  
        function func(x)  
            real    :: x  
            real    :: func  
        end function  
    end interface
```

```
    ! Variables locales
```

```
end subroutine
```

El uso de la construcción interface sirve para declarar subprogramas, funciones o subrutinas, como argumentos de otros subprogramas.

Sólo hace falta indicar el número, orden y tipo de las variables que usa el subprograma que se pasa como argumento.

Como siempre el nombre del argumento en la cabecera del subprograma no tiene porque coincidir con el nombre real usado en el principal

Programación modular: Avanzada

Implementación:

```
module funciones

contains

function f1(x)
    real :: x
    real :: f1

    f1 = exp(x*2)
end function

function f2(x)
    real :: x
    real :: f2

    f2 = cos(x)+sin(x)
end function

function f3(x)
    real :: x
    real :: f3

    f3 = x*x + 1
end function
end module
```

```
subroutine valor_maximo_intervalo(x0, x1, f1, valor_max)

    real, intent(in) :: x0,x1
    real, intent(inout) :: valor_max

    interface
        function func(x)
            real :: x
            real :: func
        end function
    end interface

    ...
end subroutine
```

```
program principal

    use subprogramas ! modulo donde esta la subrutina
    use funciones

    real :: a,b
    real :: maximo

    ...

    call valor_maximo_intervalo(a, b, f1, maximo)

end program
```


Programación modular: Ejercicio

Ejemplo

- Completad el ejercicio anterior añadiendo que la subrutina devuelva, además, el valor x donde se alcanza el máximo.
- Utilizar como base el programa anterior para calcular lo propuesto en el siguiente problema:

Sean los vectores $X, F \in \mathbb{R}^{N+1}$. Las componentes de X almacenan los valores discretos del dominio de definición y F las imágenes correspondientes de la función $F: \mathbb{R} \rightarrow \mathbb{R}$ continua a trozos siguiente:

$$F(x) = \begin{cases} 1, & a \leq x \leq -\frac{\pi}{2}, \\ \cos(\pi x), & -\frac{\pi}{2} < x < \frac{\pi}{2}, \\ 0, & \frac{\pi}{2} \leq x \leq b. \end{cases}$$

Considerar una partición equiespaciada de la forma:

$$\{x_i = a + i\Delta x, \quad i = 0 \dots N\}, \quad \Delta x = \frac{b-a}{N}, \quad a < -\frac{\pi}{2}, \quad b > \frac{\pi}{2}.$$

Se pide calcular la suma;

$$S_N = \sum_{i=0}^N F_i \Delta x$$

1. con $N = 10$
2. con $N = 20$
3. con $N = 100$